LEGAL SUFFICIENCY of TESTING PROCESSES

Clark Savage Turner, Esq., Debra J. Richardson, John L. King
Department of Information and Computer Science, University of California, Irvine
Irvine, CA., 92717 USA

Abstract

Software processes are executed for a purpose: to satisfy a set of process requirements and to meet process constraints [1]. This paper shows that there is a critical set of process constraints, often considered only implicitly or even ignored, that are derived externally from social expectations. This paper suggests an approach to determining this set of process constraints and a basic method for their consideration during safety-critical testing process design.

1 Introduction

Software is increasingly used to control systems that pose risks to the public. As dramatically demonstrated in the Therac-25 accidents [2], these risks are serious. When an innocent member of the public is injured by a new technology based on software, who bears the loss and how should software professionals deal with the issue?

Certainly, the behavior of the developer is under scrutiny. Did the development process include a reasonable set of precautions designed to avoid the predictable risks? If not, the developer might be expected to pay for the damage. If so, the developer is not at "fault" for the unfortunate accident, and the injured party may be left to suffer the damages uncompensated. This is the underlying philosophy of negligence law.

Ultimately, this discussion raises questions about externally imposed constraints on the technical requirements for safety-critical testing processes. This paper suggests that there is a set of testing process considerations that is important *beyond its ability to affect the product*, and is relevant to the long-term economic health of the organization. This set of constraints on the *process* of testing safety-critical software are imposed by the law of negligence. This paper also suggests an approach to determining a set of applicable constraints and integrating them into our safety-critical testing processes.

1.1 Why is this an important problem?

All major innovation requires investment. Investors are risk-sensitive; they avoid projects with a high risk of failure. Software engineering changes what's technically feasible while investors support the economically profitable. (See generally [3] for a thorough discussion of these issues.) Successful innovation therefore occurs at the intersection of the two worlds:

Social expectations affect the economically profitable through the law. In particular, negligence law defines a portion of the economic risk for innovative projects. The testing process may be designed to lower the risk of technical failure, but must also address the risk of economic failure in any organization with limited resources.

Detailed negligence constraints for some engineering processes are well known since they have had years to develop. Much general engineering experience and knowledge about accidents is built into these constraints. But software is new to negligence law. It not only changes the relative "size" of our projects, but their technological "reach". New social expectations and thus the applicable negligence constraints will develop in response to these changes. Our process designs must be designed to meet these constraints, and to do that, we must have an explicit set to work with!

1.2 Technical constraints on the testing process

Software testing, like negligence law, deals with the inevitable risks of new software technologies. Since no one wants a software system to arbitrarily inflict harm on innocent persons, it typically goes through a stringent testing process designed to reduce or eliminate that damage potential. It is known that in realistic situations, risks cannot be entirely eliminated by testing [4]. Thus, testing processes must attempt to minimize risk within practical limits.

Work has shown that certain processes are necessary (though not sufficient) to an effective risk reduction effort. Improvement of testing processes is an important step in the maturation of software engineering. However, evaluation and analysis of any "improvements" can only be done by explicit reference to a complete and correct set of process requirements and constraints. Therefore, it is appropriate to ask, "what are the relevant process requirements and constraints?" This paper will show that there is an important set of process constraints derived from the law of negligence.

2 The law of negligence

The tort of negligence is based upon conduct that is socially unreasonable. It involves the endeavor to "... strike some reasonable balance between the plaintiff's claim to protection against damage and the defendant's claim to freedom of action for his own ends" [5, page 6]. Negligence imposes a set of expectations on behavior through the concept of *duty*. We all have a duty to act with "reasonable prudence"[1] whenever our conduct foreseeably creates a threat of injury to others.

Negligence generally defines a portion of the economic exposure of an industry for any foreseeable damage done by its products or services whenever it has not taken reasonable precautions to avoid such damages. Since the duty in negligence focuses on behavior, it yields constraints on development processes and not the products themselves.[2] Behavior that fails to meet these process

[1] "Negligence is the omission to do something which a reasonable man, guided upon those considerations which ordinarily regulate the conduct of human affairs, would do, or doing something which a prudent and reasonable man would not do" [6].

[2] Distinguish negligence from product based liability such as strict products liability.

constraints may be a basis for negligence liability. Behavior that satisfies these constraints will not be the basis for liability, even if the product was defective and caused harm to an innocent party.[3]

The law of negligence may be different, or may be applied differently, in common law jurisdictions (like the USA and England) and civil law jurisdictions (like other countries in the European Union). Indeed, it may be applied differently within those jurisdictions, but the basic tenets remain the same, society imposes constraints on our technical processes.

2.1 Principles of negligence under common and civil law

The basic feature of the civil law that distinguishes it from the common law is that its primary source is the written law as enacted by legislators. The common law is seen to develop from principles developed through written court decisions. However, with the increasing codification of the common law through legislation in the USA and England, and the practical necessity for civil law judges to interpret cases in order to make sound decisions, the differences do not appear to be great. See generally [7]. Regardless of the source for negligence law, these constraints are derived from the same social concerns. Use of a common law model for constraint development therefore does not result in consideration of substantially different issues for the development of process constraints.[4] Therefore, the rest of this paper will assume a common law model for the development of the relevant constraints.

2.2 A common law duty to the public

In common law jurisdictions, the duty of "reasonable prudence" is determined by the court under the circumstances of the case at hand. This duty to act (or to refrain from acting) is found by a "balancing" test involving the risk[5] to the plaintiff and the burden of preventing the harm.[6] Evidence is adduced to give relative weights to the relevant factors, and if the risk of harm outweighs the cost to prevent it, then the duty is imposed. If the cost of prevention and utility to society outweigh the possible harm, then the duty is not imposed in the law of negligence.

A prominent legal scholar explains in [8] that there is no question that software engineers owe a general duty of care in negligence to their customers and the general public: *to use such care as a reasonably prudent software developer would use under similar circumstances.* What might this mean to the individual testing organization?

One cannot yet research the common law and find a list of specific software testing process constraints known to meet this general duty in negligence. We

[3] Though there may be other legal theories whereby liability is imposed

[4] Further, in an increasingly international marketplace, organizations must take note of the rules of the countries where they do business. The common law is observed in much of the international market. Any multinational organization should be aware of the common law of negligence for that reason.

[5] Risk is said to be the probability of the harm "multiplied by" the gravity of the harm.

[6] This is the burden on the defendant *and* the relative loss to society if the product has restricted utility after it is made safer.

thus develop a basic model to illustrate the process of developing constraints, giving us insight as to what they will be.

3 Legal sufficiency of testing process designs

Two things are noteworthy about the software engineer's general duty of care due under negligence law: (1) it is defined by reference to software engineering itself, and (2) perfection is not required.

3.1 The common law model

The development of specific legal rules by application of general principles to concrete situations can be modeled with a pair of interacting process control systems, where control is applied in the attempt to keep each process within certain accepted limits. See Figure 1.

Constraints on the testing process are developed through the model: the legal rule is applied in the context of technical facts and circumstances proved in court. These technical facts are derived from the field of software engineering and testing: research and accepted practices. A given constraint will develop through repeated application of the general rule to similar situations. The new constraint is established firmly by appellate review and published. It must then be faithfuly applied in subsequent cases by trial courts. It provides a public statement of the legal expectation - the constraint on the software testing process.

Use of this model to derive relevant constraints offers two main benefits to the testing process designer: (1) awareness of previously implicit (or unknown) constraints; and, (2) availability of the rich history of safety experience included in the constraints.

With this awareness, testing process designers must explicitly consider these constraints or knowingly suffer an additional risk of liability and economic loss. The general constraints may be refined given the particular circumstances existing at the time the process is to be executed. Consideration may then be given to their satisfaction and reasoned justification recorded for future use.

4 Consideration of legal process constraints

How can we explicitly include these process design constraints in our testing processes in a practical way? We suggest the use of a fault tree structure in order to trace the ways in which our testing process designs may fail to satisfy key constraints.

Fault tree analysis is a familiar tool to many involved in safety-critical systems. It is a deductive, effect - cause approach, traditionally used in hazard analysis. The fault tree is the logical model of a process with regard to some undesired event. It is represented graphically with a tree structure. See generally [9] for an industrial process approach, [10] for a software approach to fault tree construction.

Fault trees may be used to analyze our safety critical testing process designs. "Civil Liability" (money damage awards) is the "undesired top event." The

boxes indicate events, and the labels name those events. Events higher in the tree are the effects of the events lower in the tree combined with the logical operators such as AND and OR. When a box is connected to another by a single line with no logical operator, it indicates another name for the event that is critical to understanding the boundaries between the legal and software processes. Leaf nodes that are circles will indicate further development of the tree as a separate subtree with a root having the same label. These subtrees are broken off separately in order to focus on their individual properties.

The fault tree, once constructed, can be decomposed into its "minimal cut-sets" (a combination of components whose simulatneous failure is just sufficient to cause the undesired event). Note that there may be several cutsets for a given undesired top event. Each fault tree is generally not unique. The events in the trees we construct are not generally independent of each other.

4.1 The basic software process fault tree

We present a process fault tree with the undesired top event "civil liability" in Figure 2. It gives the general picture of ways civil liability may be imposed. This paper is only concerned with paths leading through a negligence node and concerned with testing, but note that in order to add overall context, other nodes and possible paths are indicated by dotted lines and boxes.

Just below the "testing process defective" node, there is an AND connector and four nodes, which represent the elements that must be proven in any negligence case. Arguably, the most critical node here is the "breach of duty" node. It is this node we choose to expand to illustrate the concepts. It is this node where the organization's activities in constraint satisfaction will be the controlling factor. The other nodes are generally legal questions that have little to do with the particular testing process design used.

It is seen that civil liability may ultimately result from either failure to reasonably document testing activities, or a failure in other testing activities. This is one way to define the "failure to test in a reasonable fashion", which is a "breach of duty" in negligence law.

4.2 Development of the testing process fault tree

The tree is developed down to its leaves that indicate individual constraints on the process. Node number 1, the reasonability of the test documentation process is diagrammed in figure 3, while node number 2, the reasonableness of other process activities is diagrammed in figure 4.

4.3 Determine the constraints

What a "reasonably prudent tester," where do we start? We may begin by looking at the literature of software testing, checking standards and industry customs to gather information about what is considered reasonably prudent testing by experts in the field. We might also check relevant legal literature to see what has been written on the topic, if anything. See, for example, [11].

We do not propose a list of potential constraints as this is beyond the scope of the present work. We choose an example to illustrate our approach to

determining and documenting reasonable efforts to consider them. In checking the testing literature, we find that one important issue is "independence." In fact, upon examination, the literature seems to be in agreement that the main testing effort for a safety-critical project should be independent of the rest of development. One very early cite to this conclusion is [12] and a recent one that speaks specifically of safety-critical projects is [13]. We further note that ISO 9000-3, though not specifically a standard for safety-critical processes, requires independence in testing and verification activities to some extent [14]. Thus, "independence" should appear as a constraint under node number 2. In any actual effort, information on all sides of the issue should be gathered and abstracted so that a well reasoned decision may be made to an approach to satisfaction or non satisfaction of the proposed constraint.

4.4 Constraint attributes

In addition to determining the minimal cutsets to determine potential problem areas of our testing processes, we suggest attribution of the leaf nodes with pointers to critical constraint information. This may be crucial to a reasoned negligence analysis during process design. It is also a foundation for defense to a future negligence suit (where we are expected to prove reasonable satisfaction of the constraints to a court of law).

The attribution to the constraints is shown in Figure 5. *External* constraint information includes all sources used in deriving the given constraint such as caselaw, statutes, regulations and other literature. It may also include indirect sources such as software engineering research, expert opinions and standards which are used in courts to derive the constraints legally.

Internal constraint information is the record of the organization's response to the given constraint. How the known risks were actually considered and resolved is shown here. Process documentation and other information relevant to cost and risk analyses used by the organization can be located from here.

With our proposed independence constraint, how might this play out? First, we must gather the information that leads us to believe that the independence constraint is important. Above, we listed citations to two references from the field of software engineering and one international standard. All the main references in addition to these should be abstracted and referenced so that they can be part of the analysis. Further, legal references that may bear on this constraint must be listed as direct or indirect authority on the constraints in question.

Second, the testing organization then needs to document its own response to the constraints with pointers to any relevant internal constraint information. To that end, the process must be documented and each part of the process that addressed the constraint in question must be noted. The particular test tools used, the extent and depth of the procedures, and the costs must be recorded for possible proof in the future. In case of any legal question, the organization would then be able to justify its own response with reasoned risk analyses, just as the court must do. If it can be shown that the organization's behavior was reasonable in light of the foreseeable circumstances, negligence liability will not follow.

5 Advantages and limitations of the approach

There are benefits to this sort of explicit approach to testing process constraints. We may increase the actual safety of the product due to a higher quality process. A safer product exposes the organization to lower risks of liability. The approach certainly heightens awareness of the constraints on our processes and may also lower the risk of lawsuits due to increased attention to them. In any case, there is a higher probability of successful negligence defense (or settlement) whenever legal action is taken, due to solid awareness of the legal issues and how these issues were handled during design. There will be lowered legal costs in any event, since the process explicitly includes relevant constraints and justifications, hence part of the lawyer's work is already done ahead of time. In some cases, this sort of approach may prevent an award of punitive damage awards by recording the organization's good faith efforts. Much of this analysis may be reusable whenever a project is shown to involve similar constraints.

This model and the approach also have limitations. Since the nodes in the fault trees are not independent, a probabilistic analysis is not contemplated. Fault tree analysis is traditionally done by hand using trained personnel. It must be done early in the process and periodically updated, since the constaints and references may change. Thus it appears to be a costly approach in the beginning of the development process. However, the potential is that the early investment will result in lower economic risks due to negligence suits. Our approach currently covers only negligence law and the testing process, but we plan to extend it to other areas in the future. Finally, this approach does not answer specific legal questions, but gives a framework for approaching a legally sound safety-critical testing process.

6 Conclusions

We have presented a model showing how legal constraints on safety-critical testing processes develop under negligence law. Even though there are no decisions published at this point for personal injury due to a software flaw, it is agreed that certain constraints may soon be found applicable to our testing processes [11], and those constraints may be postulated now and used in analysis and evaluation of our current testing processes.

We further presented an approach to analyze our safety-critical testing processes. The intent is to assure that our process designs do indeed consider a basically complete and correct set of process requirements and constraints. With the process fault trees attributed with constraint information, we can explicitly show how we addressed the relevant constraints. This allows us to later justify our particular approach to their satisfaction with the explicit consideration of key risk factors.

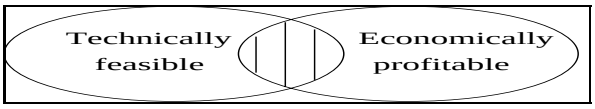


Figure 1: Common Law Process Model

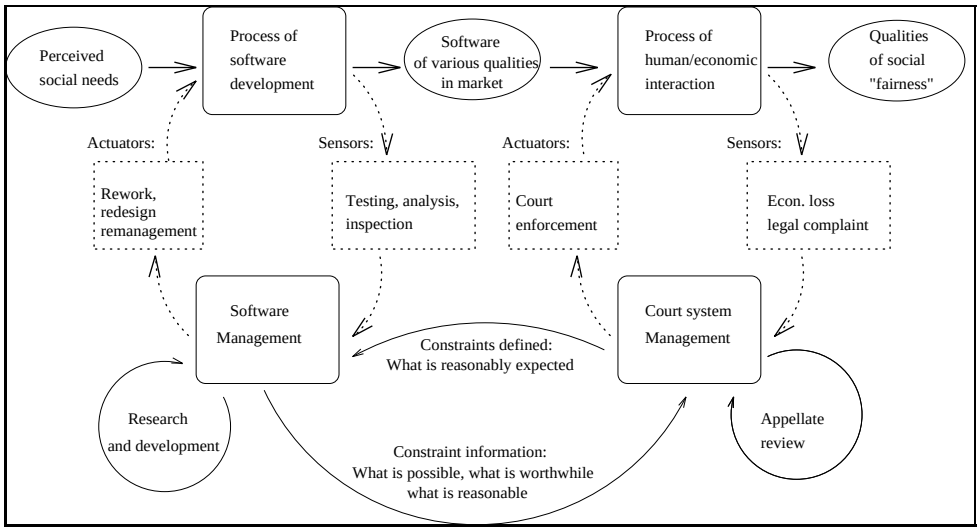


Figure 2: Basic Process Fault Tree

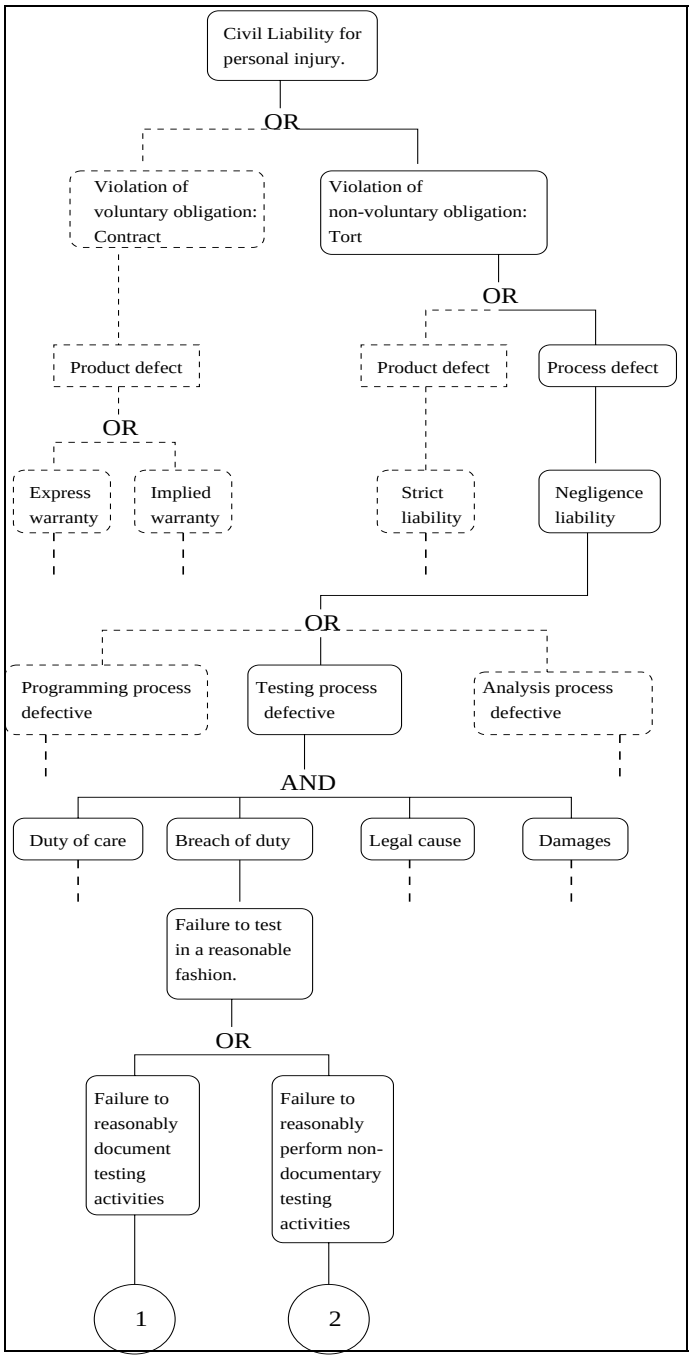


Figure 3: Documentary subtree

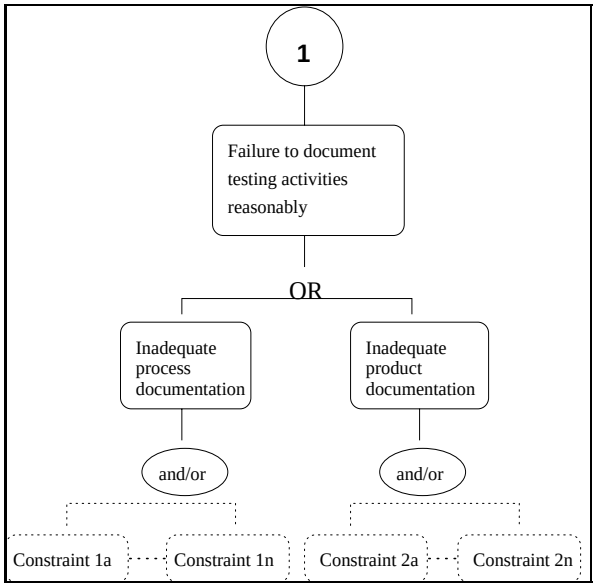


Figure 4: Nondocumentary subtree

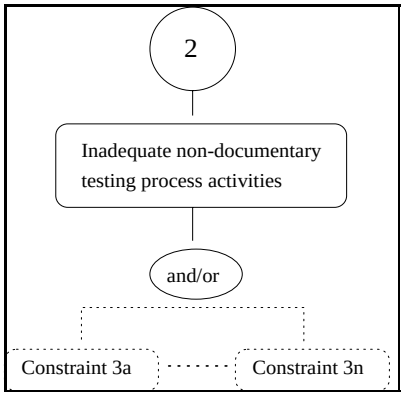
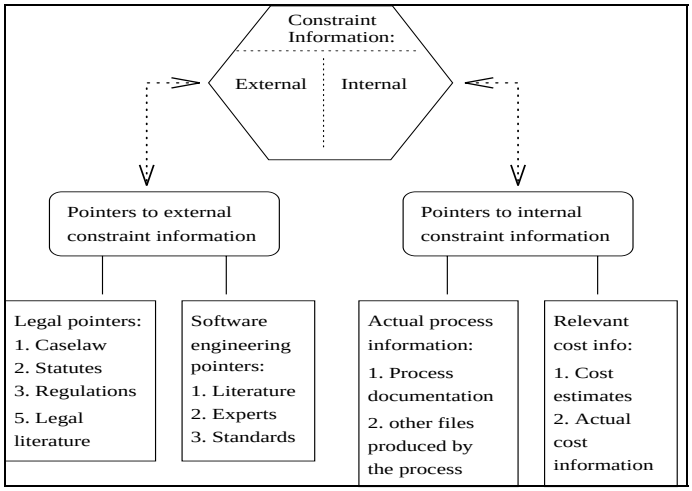


Figure 5: Constraint Attribution



References

[1] Osterweil, Leon. *Software Processes are Software Too*, 9th Int'l Conf. on Soft. Eng., 1987.

[2] Leveson, Turner, *An Investigation of the Therac-25 Accidents*, IEEE Computer, Vol 26, No. 7, July, 1993.

[3] Nelson and Winter, An Evolutionary Theory of Economic Change, Belknap Press of Harvard University Press, 1982.

[4] Hamlet, *Are We Testing for True Reliability?*, IEEE Software, July 1992

[5] Prosser, Handbook of the Law of Torts, 4th Ed., West Publ., St Paul, Minn., 1971

[6] Blyth v. Birmingham Waterworks Co., 1856, 11 Ex. 781, 784, 156 Eng. Rep. 1047.

[7] Glendon, Gordon, Osakwe, Comparative Legal Traditions, West Publ. Co., St. Paul, MN., 1994

[8] Gemignani, Law and the Computer, CBI Publ., Boston, MA., 1981

[9] Lapp, Powers, *Computer-aided Synthesis of Fault-Trees*, IEEE Trans. on Reliability, April 1977, page 2.

[10] Leveson, Harvey, Analyzing Software Safety, IEEE Trans. on Soft. Eng., Vol. SE-9, No. 5, September, 1983, page 569.

[11] Fossett, *The Development of Negligence in Computer Law*, No. Ky. L. R. 14 No. 2, pps 289-310 (1987)

[12] Weyuker, *On Testing Non-Testable Probrams*, The Computer Journal, Vol 25, No. 4, 1982

[13] Parnas, *Evaluation of Safety-Critical Software*, CACM Vol. 33, No. 6, June 1990

[14] ISO 9000-3 : 1991 (E), Quality management and quality assurance standards - Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software.